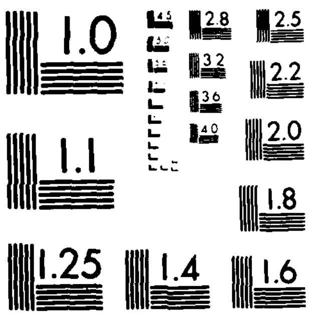
AD-A146 759 NYU ADA/ED USER'S GUIDE - VERSION 14 FOR VAX/VMS 1/1
SYSTEMS(U) NEW YORK UNIV NY COURANT INST OF
MATHEMATICAL SCIENCES 01 JUL 84 DOD/DF-85/002A
UNCLASSIFIED DAAB07-82-K-J196 F/G 9/2 NL

END
11-84
DTIC


1.0
2.8
2.5
3.2
2.2
3.6
4.0
2.0
1.1
1.8
1.25
1.4
1.6

(1)

50272-101

REPORT DOCUMENTATION PAGE	1. REPORT NO. DOD/DF-85/002a	2.	3. Recipient's Accession No.
4. Title and Subtitle NYU Ada/Ed User's Guide Version 1.4 for VAX/VMS Systems			5. Report Date 1 July 1984
			6.
7. Author(s)			8. Performing Organization Rept. No.
9. Performing Organization Name and Address Courant Institute New York University 251 Mercer Street New York, NY 10012			10. Project/Task/Work Unit No.
			11. Contract(C) or Grant(G) No. (C) DAAB07-82-K-J196 (G)
soring Organization Name and Address U.S. Army, CECOM ACTL DRSEL-TCS-ADA Fort Monmouth, NJ 07703			13. Type of Report & Period Covered
			14.

lementary Notes

This document supersedes AD-A128 708

For magnetic tape, see

act (Limit: 200 words)

This is the User's Guide for the Ada/Ed Translator, version 1.41, which was validated by the Ada Validation Facility during July 1984. Ada/Ed is the first Ada translator to be validated under version 1.4 of the ACV test suite. → to p.2

17. Document Analysis a. Descriptors

Translator
Interpreter
Ada Validation

b. Identifiers/Open-Ended Terms

c. COSATI Field/Group

18. Availability Statement:	19. Security Class (This Report) UNCLASSIFIED	21. No. of Pages
	20. Security Class (This Page) UNCLASSIFIED	22. Price

DO NOT PRINT THESE INSTRUCTIONS AS A PAGE IN A REPORT

INSTRUCTIONS

Optional Form 272, Report Documentation Page is based on Guidelines for Format and Production of Scientific and Technical Reports, ANSI Z39.18–1974 available from American National Standards Institute, 1430 Broadway, New York, New York 10018. Each separately bound report—for example, each volume in a multivolume set—shall have its unique Report Documentation Page.

1. Report Number. Each individually bound report shall carry a unique alphanumeric designation assigned by the performing organization or provided by the sponsoring organization in accordance with American National Standard ANSI Z39.23–1974, Technical Report Number (STRN). For registration of report code, contact NTIS Report Number Clearinghouse, Springfield, VA 22161. Use uppercase letters, Arabic numerals, slashes, and hyphens only, as in the following examples: FASEB/NS–75/87 and FAA/RD–75/09.
2. Leave blank.
3. Recipient's Accession Number. Reserved for use by each report recipient.
4. Title and Subtitle. Title should indicate clearly and briefly the subject coverage of the report, subordinate subtitle to the main title. When a report is prepared in more than one volume, repeat the primary title, add volume number and include subtitle for the specific volume.
5. Report Date. Each report shall carry a date indicating at least month and year. Indicate the basis on which it was selected (e.g., date of issue, date of approval, date of preparation, date published).
6. Sponsoring Agency Code. Leave blank.
7. Author(s). Give name(s) in conventional order (e.g., John R. Doe, or J. Robert Doe). List author's affiliation if it differs from the performing organization.
8. Performing Organization Report Number. Insert if performing organization wishes to assign this number.
9. Performing Organization Name and Mailing Address. Give name, street, city, state, and ZIP code. List no more than two levels of an organizational hierarchy. Display the name of the organization exactly as it should appear in Government indexes such as Government Reports Announcements & Index (GRA & I).
10. Project/Task/Work Unit Number. Use the project, task and work unit numbers under which the report was prepared.
11. Contract/Grant Number. Insert contract or grant number under which report was prepared.
12. Sponsoring Agency Name and Mailing Address. Include ZIP code. Cite main sponsors.
13. Type of Report and Period Covered. State interim, final, etc., and, if applicable, inclusive dates.
14. Performing Organization Code. Leave blank.
15. Supplementary Notes. Enter information not included elsewhere but useful, such as: Prepared in cooperation with . . . Translation of . . . Presented at conference of . . . To be published in . . . When a report is revised, include a statement whether the new report supersedes or supplements the older report.
16. Abstract. Include a brief (200 words or less) factual summary of the most significant information contained in the report. If the report contains a significant bibliography or literature survey, mention it here.
17. Document Analysis. (a). Descriptors. Select from the Thesaurus of Engineering and Scientific Terms the proper authorized terms that identify the major concept of the research and are sufficiently specific and precise to be used as index entries for cataloging.

    (b). Identifiers and Open-Ended Terms. Use identifiers for project names, code names, equipment designators, etc. Use open-ended terms written in descriptor form for those subjects for which no descriptor exists.

    (c). COSATI Field/Group. Field and Group assignments are to be taken from the 1964 COSATI Subject Category List. Since the majority of documents are multidisciplinary in nature, the primary Field/Group assignment(s) will be the specific discipline, area of human endeavor, or type of physical object. The application(s) will be cross-referenced with secondary Field/Group assignments that will follow the primary posting(s).
18. Distribution Statement. Denote public releasability, for example "Release unlimited", or limitation for reasons other than security. Cite any availability to the public, with address, order number and price, if known.

19. & 20. Security Classification. Enter U.S. Security Classification in accordance with U.S. Security Regulations (i.e., UNCLASSIFIED).

21. Number of pages. Insert the total number of pages, including introductory pages, but excluding distribution list, if any.
22. Price. Enter price in paper copy (PC) and/or microfiche (MF) if known.

NYU Ada/Ed Users' Guide
Version 1.4 for VAX/VMS systems

July 1st, 1984

A1

NOTE

The NYU Ada Translator and Interpreter has been developed by New York University for the United States Army and is made available through the Software Engineering Division of CENTACS, Fort Monmouth, New Jersey. Materials made available are solely for use at a single site; they are not to be made available to others without permission.

U.S. ARMY CECOM
CONTRACT NO. DAAB07-82-K-J196

1. History and Overview The design goal of the NYUADA project has been to produce ~~(as rapidly as possible)~~ a complete language Ada system which is faithful in all respects to the language definition, and that can serve as an operational definition of Ada, to be perused by language designers, implementors and users. The design and implementation of Ada/Ed was initiated at a point when the language was not completely defined or understood (in fact the first version was an implementation of Preliminary Ada), and has culminated in the first fully validated ANSI-Ada translator.

A conventional approach to compiler construction, realized in a medium level language such as PASCAL, BLISS or SIMULA, would have resulted in a massive program text which would not have met either criterion: usability and readability. Instead, the entire Ada system was programmed using a very high level language developed at NYU with NSF support called SETL (for SET Language). Using SETL, it was possible to create a complete Ada system in about one fifth of the time which would otherwise have been required.

There are several implementations of SETL for various computers, including the VAX 11/780 and 11/750, DEC 10/20, IBM/370, Ahmdahl/470 and CDC CYBER. Ada/Ed can be implemented on any machine running SETL, and in fact runs on the first five of the above. In any of these the SETL text of Ada/Ed, which on one hand can be viewed as a high level definition of the semantics of Ada, runs as a production program, providing compilation and execution capability for the complete ANSI Ada language.

The price paid for this very high-level approach is that NYU Ada/Ed is spectacularly inefficient. Nevertheless Ada/Ed in its current form is usable as a checkout tool, particularly for teaching and training purposes. Great effort has been put into creating a friendly interface with good error messages and diagnostics. Use of the system will tell how well this objective has been met. We believe the system establishes new standards for effective error reporting, which is particularly remarkable for a system which is entirely table driven.

On the other hand, the user will soon realize that Ada/Ed is quite unsuitable in its current form for any substantial computation unless an enormous amount of machine resources are available. The speed ratio between execution in Ada/Ed and what can be achieved in an effective production compiler is of several orders of magnitude.

Repeated references to 'the current version' will alert the reader that Ada/Ed is an evolving project, and that subsequent releases can be expected to provide better performance than this version. The current activities of the NYUADA project include the construction of a much faster interpreter, also written in SETL, and for which the current Ada/Ed system serves as a design prototype.

2. Sources of Information

2.1. Reference Material The Ada Reference Manual is available from the Government Printing Office as the Ada Language Reference Manual, ANSI / MIL-STD 1815 A, January 1983. This manual is referred to as the LRM in this document.

2.2. On-line material

A copy of each of the on line documents for the NYU Ada system may be printed on the standard system printer (queue SYS$PRINT) by typing the command line:

```
@ nyu$adaed:printdocs
```

The on line documents include:

The Ada Help file	nyu$adaed:adahelp.lis
The Ada PREDEF package:	nyu$adaed:predef.ada
The Installation Guide:	nyu$adaed:install.doc
The Users' Guide:	nyu$adaed:userman.doc
The Appendix F of the Reference Manual for this particular implementation	nyu$adaed:appendf.doc

3. Basic NYU/Ada

This chapter describes the basics of compiling and executing programs using the NYU/Ada system. It is assumed that NYU/Ada has been "installed" and that the definitions are available. The "install.doc" document in NYU$ADAED describes how to achieve this. The necessary symbol definitions and command files for using Ada are available in the file NYU$ADAED:adadefs.com . The easiest way to access them is to add:

```
$ @NYU$ADAED:ADADEFS
```

to your LOGIN.COM file.

3.1. A First Program

As a first example, we will compile and execute the primordial program which prints "Hello, world!" on the standard output. As a first attempt, we try the Ada program:

```
with text_io; use text_io;
procedure main is
begin
   putline("Hello, world!")
end main;
```

Assume that this text has been entered into a file call "test.ada" (a similar "test.ada" is provided with the Ada/Ed distribution tape). A single command compiles and executes this file:

```
$ ada test
```

and produces the following listing and statistics :

```
NYU ANSI-Ada/ED 1.4(28-May-84)        FRI 17 JUN 84  15:00:34   PAGE    1

ADAfile:     TRY.ADA
LISfile:     TRY.LIS

     1  with text_io; use text_io;
     2  procedure main is
     3  begin
     4     putline("Hello, world!")
                                   ^
*** Error: ";" expected after this token
         <------------------->
*** Semantic Error: Undeclared identifier PUTLINE (RRM 3.1)
     5   end main;

   2 translation errors detected
   Translation time: 20 seconds

$
```

Compilation takes place in two phases, call the "syntax phase" and the "semantic phase". During the first phase, the compiler detected a syntactic error on line 4 of our program - a ';' was missing after the call to putline().

During the second phase, a single error was again detected, also in line 4. The identifier "PUT-LINE" was not declared. A review of the text_io package in the Ada Reference Manual reveals that the procedure is called "put_line".

After the errors have been fixed, we try the "ada" command again and get the output:

```
NYU ANSI-Ada/ED 1.4(28-May-84)          FRI 17 JUN 84 15:01:30 PAGE     1

ADAfile:      TRY.ADA
LISfile:      TRY.LIS

    1  with text_io; use text_io;
    2  procedure main is
    3  begin
    4    put_line("Hello, world!");
    5  end;

  No translation errors detected
  Translation time: 18 seconds

  Binding time: 1.0 seconds

  Begin Ada execution

 Hello, world!


  Execution complete
  Execution time: 2 seconds
  I-code statements executed: 8
 $
```

This time, there are no syntax or semantic errors, and the compilation completes normally. The binder then goes to work and links our procedure with the text_io library, taking 1.0 seconds to achieve this. The resulting file is then interpreted, producing the output and associated statistics, the last of which indicates the number of internal code statements interpreted.

3.2. The Compilation Process Before we display another example program, some more explanation of the compilation process is in order.

In the previous section, we mentioned the two phases of the compiler: parsing and semantic analysis. At the end of these phases, a file of intermediate code, called an ais_file (for Ada Intermediate Source) is produced. If the original program appears in the file "test.ada", the corresponding ais_file will be called "test.ais". This file contains the intermediate code which the interpreter executes when the program is 'run'.

A subsequent component of Ada/Ed, namely the binder, then takes the ais_file and produces an aix_file (Ada Intermediate Executable file). The aix_file contains the intermediate code from the ais_file, bound together with the necessary compilation units from libraries requested by the program.

It is the aix_file which is passed to the interpreter for 'execution'.

The ADA command causes the compiler, binder, and interpreter to be run in succession,,and then erases the intermediate files. It is possible to retain these files, for example, if one desires to execute many times without re-compiling. Variants of the ADA command are also provided which run the various phases of the system separately. Chapter 4 describes these commands in detail.

3.3. A Larger Program For the second example, we will use a simple tasking program. Using this example we will introduce Ada libraries and the mechanism for separate compilation.

The program consists of three tasks, a producer and consumer which communicate through a buffer task. The task body of the buffer task is identical to the example in section 9.12 of the Ada Reference Manual, except that here it appears as a subunit of the main program. We will write a program which uses that separately compiled buffer task, whose source is in the file "buf.ada". Our main program is in the file "pctask.ada":

```
-- DEMONSTRATION PROGRAM:
--
--    Producer Consumer Tasking example
--
with text_io; use text_io;
procedure tasker is

task buffer is
  entry read  (c: out integer);
  entry write (c: in integer);
end;

task producer;

task consumer;

task body buffer is separate;

task body producer is
begin
  for count in 1 .. 4
  loop
    put_line("Entry call to write in buffer number: "
                & integer'image(count));
    buffer.write (count);
    put_line("Entry call to write complete.");
  end loop;
  buffer.write (0);
end producer;

task body consumer is
  use text_io;
  count: integer;
begin
  loop
    put_line("Entry call to read to get number.");
    buffer.read (count);
    exit when count = 0;
    put_line("Entry call to read obtained number: "
                & integer'image(count));
  end loop;
end consumer;

begin
  null;
end tasker;
```

The pctask.ada file must be compiled first (see LRM sect 10.3 for the rules on order of compilations) and placed in a library. To do this, we use the ADAC command which compiles a program without binding or executing. Normally, ADAC produces an ais_file, but we wish to build a library containing our two compilations. An Ada/Ed library is a specialized directory that maps the names of compilation units to the files in which their ais code reside. To construct a library we use the /LIBFILE and /NEWLIB qualifiers on ADAC:

```
$ ADAC pctask/LIBFILE=pctask/NEWLIB
```

The pctask.ada file is compiled and information to that effect is placed in a new library, pctask.lib.

Now we compile the buffer task subunit into the library by means of the command :

```
$ ADAC buf/LIBFILE=pctask
```

The library pctask.lib now contains information about the ais code for the tasker main program, the enclosed producer and consumer tasks, and the separate subunit buffer task. These may be bound and executed by:

```
$ ADAXL pctask
```

which will output:

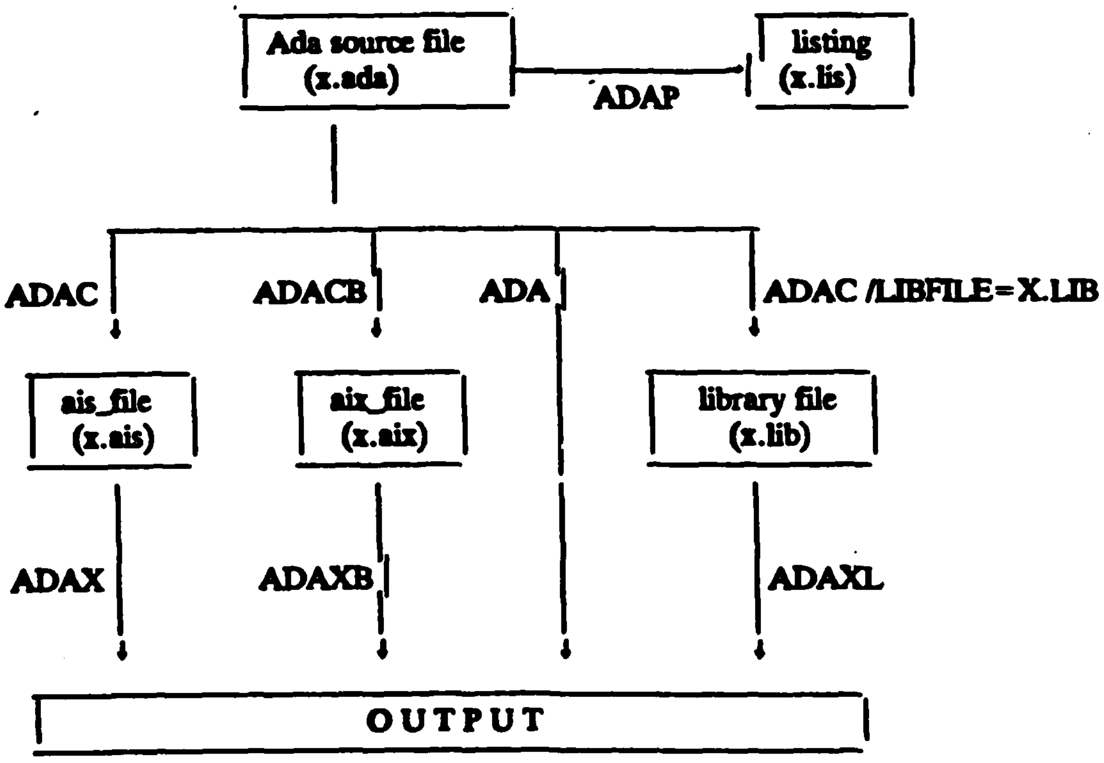
```
Binding time: 1.3 seconds

Begin Ada execution

 Entry call to read to get number.
 Entry call to write in buffer number: 1
 Entry call to write complete.
 Entry call to write in buffer number: 2
 Entry call to read obtained number: 1
 Entry call to write complete.
 Entry call to read to get number.
 Entry call to write in buffer number: 3
 Entry call to read obtained number: 2
 Entry call to read to get number.
 Entry call to write complete.
 Entry call to write in buffer number: 4
 Entry call to read obtained number: 3
 Entry call to write complete.
 Entry call to read to get number.
 Entry call to read obtained number: 4
 Entry call to read to get number.

  Execution complete
  Execution time: 99 seconds
  I-code statements executed: 151
```

As will be described below, the /AIXFILE qualifier can be added on the ADAXL command to save the pctask.aix file so that binding could have been avoided in later runs.

4. Details of the Complete System This chapter gives details of the commands available with the NYU Ada/ED system. These commands are:

ADA	compiles and executes an Ada program.
ADAC	compiles but does not execute an Ada program.
ADAX	executes a previously compiled program.
ADAXL	executes a program from a library file.
ADACB	compiles and binds an Ada program.
ADAXB	executes a compiled and bound program.
ADAP	parses an Ada program but does no semantic analysis.

These commands are summarized in the diagram:



4.1. Qualifiers This is a list of qualifiers which may be used with the commands in the NYU Ada/ED system:

/AISFILE[=file]

This qualifier indicates whether an intermediate file is to be generated. A file may be given. If the qualifier does not mention a specific file, then the intermediate file has the same name as the source file, but with the extension AIS. With ADAC the default is /AISFILE unless overridden by /NOCODE. This file may be executed by the command ADAX ais_file.

/AIXFILE[=file]

This qualifier indicates whether an executable file of bound code is to be generated. A file may be given. If the qualifier does not mention a specific file, then the executable file has the same name as the source file, but with the extension AIX. This file may be executed by the command ADAXB aix_file.

/ERRFILE[=file]

This qualifier specifies that error messages are to be listed on the specified file as they are detected. If no file is specified, the the input file name together with the default extension "ERR" will be used. If the listing file is not the default and this qualifier is not given, then the standard output file will be used as the ERRFILE. If neither the ERRFILE nor the standard listing file are specified, or if they are both specified to be the same file, then no separate error listing will be produced and errors will be listed with the source listing only. The qualifier /SL may be used to force a separate listing of errors that would otherwise be listed on the list file.

/LIBFILE[=file]

Uses the specified library to satisfy references to compilation units not present in the file being compiled (ADA or ADAC) or executed (ADA or ADAX). If an AISfile is being created (ADAC), then if it is free of errors it will be included in this library. See also the /NEWLIB qualifier.

/LIST[=file]

/NOLIST

The qualifier /LIST is used to obtain a listing of the source file. If no file is specified, then the input file name together with the default extension "LIS" will be used. If an explicit listing file is given, the extension may be omitted, in which case the default, "LIS", will be used. The qualifier /NOLIST signifies that no listing file is to be generated. If neither LIST nor NOLIST is specified for an interactive run, then the terminal is the list file -- i.e., LIST = SYS$OUTPUT. The default is /LIST.

/MAIN[=proc_name]

Specifies the name of the main program for execution or creation of an AIXfile. If the name of the main program is MAIN, then the proc_name can be omitted. When the current compilation or program library has only one main program (a parameterless procedure) this qualifier is not needed.

/NEWLIB

Indicates that a new library is to be created by the current compilation. The name of the library is indicated by the /LIBFILE qualifier. This qualifier must be specified if a library referenced by the /LIBFILE qualifier does not already exist.

/NOCODE

Overrides the default /AISFILE specification of the ADAC command, preventing the generation of an ais_file. Also overrides the default /AIXFILE specification of the ADACB command. It should be used only when only checking for compilation time errors is desired.

/NORUN

Specifies that the program is not to be executed (interpreted).

/TIO

/NOTIO (D)

Automatically prefixes every compilation unit compiled with the context specification:

```
with TEXT_IO; use TEXT_IO;
```

The default is /NOTIO.

/TRACE

Trace execution of the program.

/$SELECT=FAIR

/$SELECT=ARB

Sets the scheduling regime for open entries in a select statement.

/$SLICE=n

Set the scheduling mode (n = 0 implies FIFO, n > 0 gives a time slice).

4.2. Files If no Ada source file is given for the ADA or ADAC commands, the editor is entered and a temporary file is created. The ADA file by default has the extension "ADA", so specification of this extension is unnecessary. The default extension for other files is the three characters which preceed '_file' in the command format as described below; thus, to use the

ADAX command to execute the intermediate code file (ais_file) produced by a previous ADAC command, the extension "AIS" need not be specified as it will be supplied.

If no data_file is specified SYS$INPUT will be used for the TEXT_IO default input file. The data input file may be specified as the second parameter or with the /DATA=file_name qualifier. The extension in the file name must be given explicitly.

Either the ADA or the ADAC commands may be used to add compilation units to a library.

4.3. ADA Command The ADA command is used to compile and execute an Ada program. It should be given the name of an ada source file. If no extension to the file name is specified, "ADA" will be used. If no source file is specified then the editor will be entered to create a file (named SRC.ADA) which will be compiled and executed.

The form of the command line is:

```
ADA [ada_source_file [data_file]][qualifiers...]
```

Execution will not take place unless there are no translation errors and a main procedure is specified. The qualifier /MAIN may be used to identify the main procedure. The data_file contains the input data for the ada program. This file need not be specified if there is no input or if input is from SYS$INPUT. The data file is associated with the default input file of TEXT_IO. The ada source file may contain any number of compilation units. A library may be specified by the qualifier /LIBFILE. A bound code file may be produced for subsequent execution using the ADACB command. Use the qualifier /AIXFILE for this purpose.

4.4. ADAC Command The ADAC command is used to compile but not execute an Ada program. It should be given the name of an ada source file. If no extension to the file name is specified, "ADA" will be used. If no source file is specified then the editor will be entered to create a file (named SRC.ADA) that will be compiled.

The format of the ADAC command is:

```
ADAC [ada_source_file][qualifiers...]
```

The normal output of the ADAC command is an ais_file containing the translated form of the Ada source (use of the /AISFILE qualifier is not necessary). The translated or intermediate code can be executed subsequently using the ADAX command. The ada source file may contain any number of compilation units. A library may be specified by the qualifier /LIBFILE. The ADAC command must be used when building a library of compilation units. The library file will be updated if necessary to reflect any new units introduced by the compilation. The /AIXFILE qualifier may be used to produce a bound program if a main procedure and all required library units and subunits are present.

4.5. ADACB Command The ADACB command is used to compile and bind but not execute an Ada program. It should be given the name of an ada source file. If no extension to the file name is specified, "ADA" will be used. If no source file is specified then the editor will be entered to create a file (named SRC.ADA) that will be compiled and executed.

The format of the ADACB command is:

```
ADACB [ada_file][qualifiers...]
```

If there are no translation errors, the ADACB command produces a bound translated program in an aix_file. Use of the /AIXFILE qualifier is not necessary. There must be a unique main procedure (parameterless). The name of this procedure may be specified by the /MAIN qualifier. The ada source file may contain any number of compilation units. A library may be specified by the qualifier /LIBFILE.

4.6. ADAX Command The ADAX command is used to execute a previously compiled program. The input file must be an AISfile produced by a previous ADAC or ADA command. If no extension is given, a default extension of "AIS" will be used. A LIBFILE must also be supplied if any compilation units from the library are needed. The input file must contain a single main program (parameterless procedure). This may be specified with the /MAIN qualifier.

The format of the ADAX command is:

```
ADAX ais_file [data_file][qualifiers...]
```

Additionally the qualifier /AIXFILE may be used to produce a bound code file.

4.7. ADAXB Command The ADAXB command is used to execute code in an AIXfile. The input file must be an AIXfile produced by a previous compilation using the ADACB command or the /AIXFILE qualifier. If no extension is given a default extension of "AIX" will be used. The bound code in that file is executed by the interpreter.

The format of the ADAXB command is:

```
ADAXB aix_file [data_file][qualifiers...]
```

4.8. ADAXL Command The ADAXL command is used to execute a program whose compilation units are given in a Library file. The input file is the library that contains the main program and its dependent units. If no extension is given, the default of "LIB" will be used. The /MAIN qualifier must be used if there is more than one parameterless procedure in the library that could be executed as the main procedure. The program that results from binding of the library units is then executed. The /AIXFILE may be used to save the bound program for later execution. An input data file may be specified as the second argument to the command or by using the /DATA qualifier.

The format of the ADAXL command is:

```
ADAXL lib_file [data_file][qualifiers...]
```

4.9. ADAP Command The ADAP command is used to parse an Ada source program. It may be used to check a program for proper syntax before compiling it.

The format of the ADAP command is:

```
ADAP [ada_file][qualifiers...]
```

The appropriate qualifiers are /ERRFILE, to specify an error file and /LIST to specify a listing file.

4.9.1 adaprs Command

The *adaprs* command is used to parse an Ada source program. It can be used to check a program for proper syntax and limited semantics before compiling it with *adac*. This command is equivalent to the *adap* command, however it is an order of magnitude faster. *Adap* is still available and can be used if *adaprs* does not perform properly on a given program. Please report such problems to the NYU / Ada Project.

The format of the *adaprs* command is:

```
adaprs ada_file [[-nl] | [-l[listing_file]]]
```

-nl = Do not produce a listing file as indicated
-l = Produce a listing file, optionally specifying it

If the ada_file name is in the format of xxxxxx.ada and the -l parameter is used the listing_file will have the name xxxxx.lis. If the -l is not specified the listing will be displayed on the terminal.

```
$ adaprs test.ada
The listing file is displayed on the terminal

$ adaprs test.ada -l
The listing file is created as test.lis

$ adaprs test.ada -ltest.ls2
The listing file is created as test.ls2 .

$ adaprs test.ada -nl
No listing file is produced
```

An auxillary file xxxxx.msg is produced after each use of *adaprs* and may be ignored.

5. I/O As part of the support environment for Ada, the TEXT_IO package is available for accessing and manipulating character files on a line oriented basis. In addition the packages SEQUENTIAL_IO and DIRECT_IO are available to perform I/O on any kind of internal Ada objects. The exceptions that may be raised while performing I/O are defined in the IO_EXCEPTIONS package. The specification of all these packages is given in chapter 14 of the Reference Manual.

5.1. TEXT_IO A package for performing Input/Output on files of characters as defined in the reference manual is available under the name TEXT_IO.

The qualifier /TIO supplied when a compilation is being made (ADA and ADAC commands) will automatically prefix every compilation unit in the source file with the context:

```
with TEXT_IO; use TEXT_IO;
```

See section 4.1.

When using TEXT_IO the standard input file is the input data file (by default SYS$INPUT). If this file is the user's terminal, a prompt will be issued whenever a line of input is required. The prompt character is '>'. On output no line is written until it is complete. Thus, PUT(X), does not write X out immediately, but places it in an output buffer. The buffer is flushed when a PUT_LINE or NEW_LINE call is made or when the program has finished execution. Thus, do NOT use PUT with a STRING argument as a prompt or debugging aid; use PUT_LINE instead.

5.2. File management under VAX/VMS

See Appendix F : Implementation Dependent Characteristics at the end of this user's manual.

6. Using Libraries To create a Library use the /LIBFILE=library_name and /NEWLIB qualifiers when compiling the first compilation unit of that library. If no extension is specified for the library file name, "LIB" will be supplied by default.

To add a compilation to an existing Ada library simply compile it using the ADA (or ADAC) commands in the presence of the library (given by the /LIBFILE qualifier).

For example:

```
! creates lib w/ part1.ais
ADAC part1/LIBFILE=libr/NEWLIB

! adds part2.ais to libr.lib
ADAC part2/LIBFILE=libr

! adds part3.ais to the library
ADAC part3/LIBFILE=libr

! executes the main procedure in libr.lib
ADAXL libr/MAIN=unit_name
```

7. Error Diagnostics There are four classes of error messages: Lexical, Syntactic, Semantic, and Execution. Messages of the first three types may arise during compilation; Execution errors occur only during interpretation. The form of an error message is illustrated by:

```
        FOR I FROM 1 TO 10 LOOP
        <-----^^^^^^--->
    *** Syntactic Error: unexpected input ignored.
```

The message gives the error class followed by some explanatory text. For Lexical, Syntactic, and Semantic errors, the portion of text in error is indicated on the listing and error files. This is done when possible by underlining the incorrect text as illustrated above. For syntax errors the token at which the error was discovered may be highlighted by circumflexes. Thus, the token "FROM" is marked as the error point. Note that tokens to the left and right of the error token may be dropped by the parser, in attempt to recover from the error. The semantic error messages are for the most part self-explanatory.

Execution errors raise the exception SYSTEM_ERROR, defined in package SYSTEM. The programmer may write handlers for this exception. When an exception is raised and there is no handler provided for it, the following form of message may be produced :

```
*** Exception END_ERROR raised in MAINTASK MAIN  statement 5
*** No handler, task is terminated
```

On the other hand, an error message of the form:

```
  *** ERROR AT STATEMENT ...
```

signifies an internal compiler error. Please report such errors.

If a program contains a large number of errors, the compiler may, on very rare circumstances, abort before termination and without producing any listing. In such cases, rerun the compiler with the /ERRFILE qualifier. The error file generated will contain all error messages produced before the point of the compiler termination.

LIST OF SEMANTIC ERROR MESSAGES

The following is the list of semantic errors that can be emitted by Ada/Ed. Any text between double quotes is replaced by an actual identifier or expression. The numbers at left refer to the Reference Manual paragraphs which describe the error condition being diagnosed.

2.8	Invalid pragma format
3.6.1, 3.7.2	Unconstrained "array or record" in object declaration
3.2	Missing initialization in constant declaration
3.2, 7.4	Invalid context for deferred constant
3.2, 7.4	"id" is not a deferred constant
8.3	Invalid redeclaration of "id"
7.4, 7.4.1	incorrect type in redeclaration of "id"

7.4	Missing initialization in redeclaration of "id"
7.4.4	Initialization not available for entities of limited type
7.4.3	premature use of deferred constant before its full declaration
3.2	Expect literal expression in number declaration
3.2	Expect universal numeric type in number declaration
3.7.1	Invalid use of discriminants
8.3	Invalid redeclaration of "id"
3.8	Incomplete type definition must be completed in the same scope in which it first appears
7.4	Invalid context for redeclaration of private type
12.1	Generic private type "id" cannot have declaration in private part
3.3	Invalid type_mark in subtype indication
3.3	invalid use of type "id" within its own definition
3.8	Invalid constraint on access type
3.6.1, 3.7.2	Invalid subtype indication: type is already constrained
3.3, 3.6.1	Invalid type mark in subtype indication: "id"
3.4	A derived or private type defined in a visible part cannot be derived in the same visible part
3.4	cannot obtain derived type from "id"
3.5.4	bounds in an integer type definition must be static
3.5.4	bounds in an integer type definition must be of some integer type
3.5.7	Expect static expression for digits
3.5.7	Expect integer expression for DIGITS
3.5.7	Invalid digits value in real type declaration
3.5.7	Expect static expression for delta
3.5.9	Expression for delta must be of some real type
3.5.9	Missing range in Fixed type declaration
3.5.7, 3.5.9	Bound in range constraint of type definition must be static
3.5.7, 3.5.9	Bound in constraint of real type declaration must be a real type
3.3, 3.6.1	Invalid RANGE constraint for type
3.3	Invalid constraint for type
12.1	accurracy constraint cannot depend on a generic type
3.5.7	Expect static expression for DIGITS
3.5.7	Invalid value for DIGITS
3.3	Invalid expression for range constraint
3.3	RANGE attribute has wrong type for constraint
3.3, 3.6.1	Invalid subtype name in constraint
3.3	invalid constraint for type
3.6.1	Constraints apply to all indices or none
3.6.1, 3.7.2	Unconstrained element type in array declaration
3.6.1	Array type is already constrained
3.6.1	Incorrect no. of index constraints for type "id"
3.3, 3.6.1	Type constraint error
3.6.1	Invalid type mark in subtype definition "id"
3.6.1	Invalid expression for index definition

3.3, 3.6.1	Invalid bounds expressions in range
3.3, 3.6.1	expect discrete type in discrete range
3.6.1	Invalid index constraint for "id"
3.7.1	Incomplete specification of default values for discriminants
3.8 , 7.4.1	Discriminants do not match previous declaration
3.7.1	Discriminant must have discrete type
3.1	Invalid self-reference in definition of "id"
3.6.1, 3.7.2	Unconstrained "array or record" in component declaration
3.3, 3.7.2	Invalid type for constraint
3.7.1, 3.7.2	Invalid constraint: Record type has no discriminant
3.7.2	Positional associations after named ones
3.7.2	Too many constraints for record type,
3.7.1, 3.7.2	Expect discriminant names only in discriminant constraint
3.7.1, 3.7.2	Invalid discriminant name in discriminant constraint
3.7.1, 3.7.2	Duplicate constraint for discriminant "id"
3.7.2	Missing constraints for discriminants
3.7.1	a discriminant appearing in a subtype indication must appear by itself
3.7.1, 3.7.3	Invalid discriminant name in variant part
3.7.1, 3.7.3	invalid redeclaration of "id" in private part
3.8, 8.2	invalid redeclaration of "id"
4	"expr" has incorrect type.Expect "id"
4.8, 3.8	Context of allocator must be an access-type
3.8, 4.8	Invalid type for allocator. Expect "id"
4.6	ambiguous expression for conversion
4.6	Invalid array conversion
4.6	conversion not possible to type "id"
4	Incorrect type for expression. Expect "id"
3.5.9, 4.5.5	Missing explicit conversion from universal fixed value
6.4	Invalid argument list : positional parameter after named parameter
4.5.5, 4.10	Invalid context for mixed mode operation
4.5.5	Missing explicit conversion from universal_fixed value
7.4.2	"op_name" not available on a limited type
6.4.1	"mode" actual parameter no. "no" in call is not a variable
4.3	OTHERS must be the last aggregate component
4.3.1	No value supplied for discriminant "id"
4.3.1	Value for discriminant "id" must be static
3.7.2	Discriminant value for "id" is out of range
4.3.1	undefined component name "id"
4.3.1	Duplicate value for component "id" in aggregate
4.3.1	Range choice not allowed in record aggregate
4.3.1	OTHERS choice must represent at least one component
4.3.1	OTHERS expression incompatible with "id"
4.3.1	components on a choice list must have same type
none	Too many components for record aggregate
4.3.1	No value supplied for component "id"
3.5.9, 4.5.5	Missing explicit conversion from universal_fixed value
6.6, 8.3	Ambiguous call to one of "id","id"...
6.7, 8.3	Ambiguous operands for "op"

3.5.1, 4.7, 8.3	Ambiguous literal: "expr"
8.2, 8.3	ambiguous expression
8.4, 14.4	Text_io not instantiated nor defined for type
3.5.1	no instance of "opn" has type "id"
none	Expect expression to yield type "id"
7.4.2	assignment not available on a limited type
5.2	left-hand side in assignment is not a variable
3.7.3, 5.4	Case expression not of discrete type
5.4	Case expression cannot be of a generic type
3.7.3, 5.4	The choice OTHERS must appear alone and last
5.4	choice must have type "id"
3.7.3, 5.4	Case choice not static,
5.4	choice value(s) not in range of static subtype of case expression
3.7.3, 5.4	Duplicate choice value(s),
3.7.3, 5.4	Missing OTHERS choice
5.6	invalid block id. "id"
5.6	missing block id. "id"
5.7	EXIT statement not in loop
5.5, 5.7	Invalid loop label in EXIT: "id"
5.7	attempt to exit from "entity"
5.8	invalid context for RETURN statement
5.8	Procedure cannot return value
5.8	Function must return value
5.1	Duplicate identifier for label: "id"
5.1	Undefined label
5.9	Unreacheable label: "id"
6.7	Incorrect no. of arguments for operator "id"
6.7	Initializations not allowed for operators
6.7	Parameters of = must have same type
6.7	= can only be defined for limited types
6.7	/= cannot be given an explicit definition
6.1	Duplicate formal parameter: "id"
6.5	in_out parameter not allowed for functions
6.1	default initialization only allowed for IN parameters
7.4.2	Default initialization not allowed for limited types
6.3.1	Declaration does not match previous specification
5.1	Unsuported feature: not procedure or entry call
6.5, 9.5	Invalid procedure or entry call
5.1	Invalid statement
9.5	Invalid call,
4.5	"desig" is not an operator designator
8.3, 8.4	Redeclaration of identifier: "id"
6.6	Redeclaration of subprogram: "id"
7.1, 9.1	specification and body are in different scopes
7.1, 9.1	Matching specification not found for body "id"
7.4, 12.1.2	Invalid context for private declaration
8.2	Invalid redeclaration of "id"
3.8.1, 7.4.1	invalid use of type "id" before its full declaration
7.4	Wrong scope for type of deferred constant
7.4.1	Private type cannot be fully declared with a type without assignment
7.4.1	Private type cannot be fully declared as an

	unconstrained array type
7.4.1	Private type without discriminants cannot be given full declaration with discriminants
7.4.1	Missing redeclaration in private part for "id"
7.2 , 7.4.1	Redeclaration of "id" in private part
7.3	Missing body for "proc or func" "id"
3.8.1	Missing full type declaration for incomplete type "id"
10.1	Compilation unit not found: "id"
3.1	Undeclared identifier "id"
3.3	Invalid reference to type "id"
8.3, 8.4	Ambiguous identifier. Could be one of: "id","id"...
3.2	Invalid object for name: "id"
4.1.3, 8.3	selector "id" not declared in "id"
4.1.3	Invalid name for selected component: "id"
4.1.3	Ambiguous name in selected component: "id"
8.3	duplicate identifier: "id"
8.5	Expect identifier in renaming of "kind"
8.5	"id" is not "kind"
8.5	Object cannot be renamed as procedure
8.5	Object cannot be renamed as function
4.1.4, 8.5	Expect a type for the attribute
8.5	Function spec. does not match renamed attribute
8.5	invalid type_mark in RENAMES clause
8.5	existence of object "id" depends on a discriminant
8.5,12.3.6	- ambiguous subprogram name: "id"
8.5,12.3.6	*No match for subprogam specification*
9.5	Cannot rename entry family as a whole
9.5	Invalid index on an entry (not entry family)
8.5	No entry match for subprogram specification
8.3	Duplicate declaration of "id"
8.4, 10.1	undeclared name in USE list "id"
8.4	"id" is not the name of a USEable package
9.5	Accept statements can only appear in tasks
9.5	Undefined entry name in ACCEPT: "id"
9.5	invalid index on entry (not entry family)
9.5	ACCEPT statement does not match any entry
9.5	Missing index for entry family
9.5	invalid entry name in accept: "id"
9.5	An accept statement cannot appear within an accept for the same entry
9.5, 3.6.1	Call to member of entry family requires one index
9.5	invalid index. "id" is not an entry family
9.7.2	expect entry name in entry call
9.5	Missing index in name of member of entry family
9.5, 3.6.1	Member of entry family requires a single index
9.5	Expect task object in entry name
,	Expect task object, not type
9.5	Undefined entry name in task : "id"
9.7.1	Delay and terminate alternatives cannot appear in the same SELECT statement
9.7.1	ELSE part cannot appear in SELECT statement if delay or terminate alternatives are present
9.7.1	Only one terminate alternative can appear in a SELECT

	statement
9.7.1	SELECT statement must have at least one ACCEPT alternative
9.10	Expect task object in abort statement
9.10	Invalid task type in ABORT statement
10.1.1	Unknown unit in with clause: "id"
7.1, 9.1	specification and stub for "id" are in different scopes
7.1, 9.1	Matching specification not found for stub "id"
none	Invalid compilation unit
10.1	Name of separately compiled unit cannot be an operator designator
10.2	Subunit identifier not unique: "id"
10.2	stubs can only appear in the outermost scope of a compilation unit
10.2	cannot find stub for subunit "id"
10.2	cannot find stub for subunit "id"
11.2	OTHERS must appear last in handler
11.1	"id" is not an exception
11.2	duplicate exception name in handler
11.2	Duplicate OTHERS in exception part
11.2	OTHERS must appear alone in exception list
11.3	RAISE statement not directly in exception handler
8.3	Duplicate declaration of "id"
12.1.1	Type of a generic formal object of mode IN must not be a limited type
12.1.1	Initialization not allowed for IN OUT generic parameters
12.1.1	OUT generic formals objects not allowed
12.1.2	constants of a generic type cannot be deferred
6.7,12.3.1	explicit overloading of the inequality operator is not allowed
12.1, 12.3	"id" is not a generic "kind"
12.1	"id" is not a generic package
12.3	Recursive instantiation not allowed
12.3	Too many actuals in generic instantiation
12.3	Positional association after named one
12.3	Missing instantiation for generic parameter "id"
12.3	duplicate or erroneous named associations in instantiation
12.1.1, 12.3.1	Instantiation of generic in out parameter is not a variable
12.3.1	Instantiation of generic in out parameter cannot be a conversion
12.3.1	existence of generic in out parameter "id" depends on a discriminant
12.3, 12.3.2-12.3.5	invalid expression for instantiation of "id"
12.3	invalid expression for instantiation of "id"
12.3.2-12.3.5	Expect type to instantiate "id"
12.3.2 - 12.3.5	Invalid type for instantiation of "id"
12.3	Invalid use of incomplete type in instantiation of "id"
12.3	Invalid use of private type in instantiation of "id"
12.3.2	Expect non-limited type to instantiate "id"
12.3.2	discriminant mismatch in instantiation of "id"
12.3.2	Instantiation of "id" must be unconstrained

12.3.2	Usage of private type "id" requires instantiation with constrained type
12.3.3	expect access to "id" to instantiate "id"
12.3.3	formal and actual designated types must be both constrained or unconstrained
12.3.5	Expect access type to instantiate "id"
12.3.4	Expect array type to instantiate "id"
12.3.4	Expect constrained array type to instantiate "id"
12.3.4	Expect unconstrained array type to instantiate "id"
12.3.4	Dimensions of actual type do not match those of "id"
12.3.4	index or component type mismatch in instantiation of array type "id"
12.3.4	formal and actual array component type must be both constrained or unconstrained
12.3.6	ambiguous or invalid match for generic subprogram "id"
12.3.6	Attribute does not match generic subprog.sp pec
12.3.6	Invalid object for instantiation of generic subprog
13.1	type "id" does not appear in same declarative part
7.4.1	type "id" appears before its full type declaration
13.2	Expression in size spec is not static
13.2	Prefix of attribute is not type or first named subtype
13.2	Prefix of attribute is not task type or access type
13.3	Identifier is not an enumeration type
13.3	Integer code is not distinct or violates predefined ordering relation of type
13.3	Component of aggregate in enumeration representation clause is not static
13.4	Identifier is not a record type
13.4	Alignment clause must contain a static expression
none	Component "id" does not appear in record type
none	Component "id" already occurs in clause
13.4	Expression for component "id" must be static
13.4	Range for component "id" must be static
3.3	Invalid expression for range constraint
3.3	RANGE attribute has wrong type for constraint
13.7.2, Annex A	use of SYSTEM.ADDRESS requires presence of package SYSTEM
3.6.2	Invalid or ambiguous argument for attribute RANGE
3.6.2	attribute "id" is undefined on unconstrained type "id"
4.3, 4.3.2	OTHERS choice not allowed in this context
4.3.2	In a positional aggregate only named association allowed is OTHERS
4.3.2	Named aggregate choice not static
4.3	Aggregate contains duplicate choices
4.3	Missing values in aggregate
Appendices B,F	Format error in pragma "id"
Appendix B	Priority must be static
Appendix B	Invalid format for pragma priority
	Invalid pragma format
	Invalid file name
Appendices B,F	format error in pragma "id"
none	System error: invalid node type "node"
none	"feature" is not supported in current implementation

none	Precision not supported by implementation
none	Implementation only supports digits 6
none	system error: strange op type "id"
none	system error : empty label stack

8. Restrictions and Limitations There are no set limits on the number of identifiers, lines of code, number of procedures, packages, or tasks that may appear in a program. Practical limitations are imposed only by the size of the total memory available. In general, programs of more than a thousand lines will probably exhaust storage, and should be compiled in several separate units if possible.

The following miscellaneous limitations should be observed by the user of Ada/Ed Version 1.4:

(1) A generic object cannot contain any stubs.

(2) A compilation can refer only to a single library.

(3) Representation specifications are not implemented.

9. Appendix F: Implementation Dependent Characteristics

(1) The form, allowed places, and effect of implementation dependent pragmas.

NYU Ada/Ed does not recognize any implementation dependent pragmas. The language defined pragmas are correctly recognized and their legality is checked, but, with the exception of LIST and PRIORITY, they have no effect on the execution of the program. A warning message is generated to indicate that the pragma is ignored by Ada/Ed.

(2) The name and the type of every implementation dependent attribute.

There are no implementation dependent attributes in NYU Ada/Ed.

(3) The specification of the package system.

```
package SYSTEM is

  type NAME       is (ADA_ED);
  type ADDRESS    is new INTEGER;

  SYSTEM_NAME    : constant NAME := ADA_ED;
  STORAGE_UNIT   : constant := 32;
  MEMORY_SIZE    : constant := 2**30 - 1;

  -- System Dependent Named Numbers:

  MIN_INT        : constant := -(2**30 - 1);
  MAX_INT        : constant := 2**30 - 1;
  MAX_DIGITS     : constant := 6;
  MAX_MANTISSA   : constant := 1000;
  FINE_DELTA     : constant := 2.0 ** (-30);
  TICK           : constant := 0.01;

  -- Other System Dependent Declarations

  subtype PRIORITY is INTEGER range 0 .. 9;

  SYSTEM_ERROR  : exception;
              -- raised if internal check fails

end SYSTEM;
```

(4) The list of all restrictions on representation clauses.

NYU Ada/Ed does not support any representation clauses and a program containing any instance of a representation clause is considered to be illegal.

Representation attributes are recognized, as required by the definition of the language, but always return a value of zero, since concepts such as number of bits and address do not apply to the implementation.

(5) The conventions used for system generated names.

NYU Ada/Ed does not provide any system generated names denoting system dependent entities, since in any case, representation specifications are not permitted.

(6) The interpretation of expressions that appear in address clauses.

Address expressions in NYU/AdaEd are meaningless, since the model used for interpretation does not use addresses. The ADDRESS type defined in SYSTEM is present only for completeness, and to be able to recognize semantically legal uses of the attribute ADDRESS.

(7) Restrictions on unchecked conversion.

NYU Ada/Ed will correctly recognize and check the validity of any use of unchecked conversion. However, any program which executes an unchecked conversion is considered to be erroneous, and the exception PROGRAM_ERROR will be raised.

(8) Implementation dependent characteristics of the input-output package.

A) Temporary files are fully supported. The naming convention used is as follows:

XHHMMSS.TMP

X stands for the file accessing method
S - SEQUENTIAL_IO
D - DIRECT_IO
T - TEXT_IO

HH - hour of file creation
MM - minute of file creation
SS - second of file creation

B). Deletion of files is fully supported.

C) Only one internal file may be associated with the same external file (No multiple accessing of files allowed).

D) File names used in the CREATE and OPEN procedures are standard VMS file names. The function FORM returns the string given as FORM parameter when a file is created. No system-dependent characteristics are associated with that parameter.

E) A maximum of 17 files can be open at any given time during program execution.

F) The standard default input file may be specified using the DATA parameter of the ADA commands. If a file is specified it must be possible to open it at the beginning of program execution, otherwise the exception PROGRAM_ERROR will be raised. If no file is specified SYS$INPUT will be used. The standard output file is SYS$OUTPUT.

G) SEQUENTIAL_IO and DIRECT_IO support constrained array types, record types without discriminants and record types with discriminants with defaults.

H) I/O on access types is possible, but usage of access values read in another program execution is erroneous.

I) Normal termination of the main program causes all open files to be closed, and all temporary files to be deleted.

J) LOW_LEVEL_IO is not supported.

K) The form feed character (CTRL L - ascii 12) is used as the page terminator indicator. Its use as a data element of a file is therefore undefined.

ADA/ED SOFTWARE TROUBLE REPORT

VERSION NO.	DATE	USER ID NO.(LEAVE BLANK)	TROUBLE REPORT NO.

ATTACHMENTS INCLUDE

___ SYSTEM ERROR MESSAGE	___ SOURCE LISTING	___ DOCUMENT(S)
___ INPUT DATA	___ COMPILED LISTING	___ OTHER ___________
___ OUTPUT DATA	___ MAGNETIC MEDIA	___________

USER NAME/DIVISION/COMPANY/ADDRESS/ CITY, STATE, ZIP CODE	VAX 11/780 CONFIGURATION
	CRT TERMINAL TYPE ___________
	MAIN MEMORY SIZE ___________
	WORKING SET SIZE ___________
	OPERATING SYSTEM/VERSION ______
(AREA CODE) PHONE NAME	

PROBLEM DESCRIPTION

USER RECOMMENDATION

SOFTWARE TECHNOLOGY DEVELOPMENT USE

ACTION TAKEN

NAME	OFFICE	PHONE	INITIALS	DATE

REVIEWER	OFFICE	PHONE	INITIALS	DATE

STATUS: OPEN CLOSED

US ARMY - CECOM
Software Tech Dev Div, CENTACS
ATTN: DRSEL-TCS-ADA-1 (T. BUCK)
FT. MONMOUTH, NJ 07703-5204

Ada/Ed User Data

Company Name ______________________________

Address ______________________________

City/State/Zip Code ______________________________

User Name ______________________________

Division ______________________________

Phone ______________________________

Version ______________________________

VAX Configuration:

Model ____________________

Main Memory Size (Mb) ____________

Working Set Size (pgs) ____________

Operating System/Version ____________

Internal Use Only:

Date Copied ____________________

Method of Request ____________________

Method of Delivery ____________________

Date Mailed ____________________

Send To:

Commander, CECOM
Software Tech Dev Div, CENTACS
ATTN: DRSEL-TCS-ADA-1 (T. Buck)
Fort Monmouth, NJ 07703-5204

New York University
A private university in the public service

Courant Institute of Mathematical Sciences
Department of Computer Science

251 Mercer Street
New York, N.Y. 10012
Telephone: (212) 460-7100

May 18, 1983

To the attention of:
Mr. Walter Finch
NTIS
5285 Port Royal Road
Springfield, Virginia 22161

To Whom it May Concern:

The NTIS is authorized to reproduce and sell this copyrighted work:

Ada/Ed users guide

Permission for further reproduction must be obtained from the copyright owner.

Sincerely,

Edmond Schonberg

Dr. E. Schonberg
NYU/ADA project

-8